Scaling Virtual Network Functions

Scaling Virtual Network Functions Using ETSI API

One of the main benefits of ESC is its capability to elastically scale a service. This allows a VNFC that performs a particular role or aspect within the VNF to be able to service requests and scale out to meet high demand or scale in when being under utilized. This aspect may span across multiple VNFCs.

The scaling requests may be manual or automatic. The different approaches to accomplishing scaling are detailed below.

For more details on these concepts and specification, please see Annex B of *ETSI GS NFV-SOL 003*.

For information on Scaling VNFs using REST and NETCONF APIs, see the *Cisco Elastic Services Controller User Guide*.

Scale

The Scale VNF request uses the *scaleStatus*, an attribute found as part of the instantiatedVnfInfo when querying a VnfInstance resource. This attribute describes the current scale level of each aspect in the VNF, for example:

```
"scaleInfo": [
  {
    "aspectId": "webserver", "scaleLevel": "4"
  },
  {
    "aspectId": "processing", "scaleLevel": "2"
  }
]
```

This forms the starting point for a Scale VNF request, which allows a single aspect to be scaled horizontally (i.e. adding or removing VNFCs) relative to the current *scaleLevel* for that dimension of the VNF. Any scaling operation on an aspect will be applied to each VNFC that supports that aspect.

Note The current specification does not support vertical scaling (adding/removing resources to/from existing VNFC instances) at this time.

Request Payload (ETSI data structure: ScaleVNFRequest)

```
{
    "type": "SCALE_OUT",
    "aspectId": "processing",
    "numberOfSteps": 1,
    "additionalParams": {}
}
```

The above payload results in the *scaleStatus* example above being updated to and the addition of the number of VNFCs for this step required to scale out to scaleLevel 3:

```
"scaleInfo": [
  {
    "aspectId": "webserver", "scaleLevel": "4"
  },
  {
    "aspectId": "processing", "scaleLevel": "3"
  }
]
```

To understand the scaling steps and other related policies configured to support scaling, see the VNFD Policies for Scaling.

Scale To Level

The Scale VNF To Level request, rather than the relative scaling that Scale VNF offers, specifies the absolute scale result desired and so some aspects may be scaled out and others scaled in. This option uses one of the two approaches to define the scaling required:

- instantiation level
- scale level

These are mutually exclusive and allow for more than one aspect to be scaled in a single request.

Instantiation Level

An Instantiation level is a predefined size for each aspect, where each level has a scale level associated with each aspect. There is no further granularity offered and so the entire VNF (that is, all aspects) is scaled according to the instantiation level requested.

Example:

Request Payload (ETSI data structure: ScaleVNFToLevelRequest)

```
{
    "instantiationLevelId": "premium"
}
```

See the VNFD Policies for the definition of instantiation levels.

Scale Level

The Scale Level is also a pre-defined size for each aspect where each aspect has target VNFCs, defined step_deltas (since each scaling step may not be uniform) and a maximum scale level. The policies that define this option allow the different targets to have different scaling outcomes.

Note The scale level does not represent the number of VMs; for example scaleLevel=0 means the initial number of instances (initial delta) for that aspect on the target VNFC and scaleLevel=1 is the initial delta plus the first scaling step defined for that aspect and VNFC tuple.

Request Payload (ETSI data structure: ScaleVNFToLevelRequest)

```
{
    "scaleInfo": [
      {
            "aspectId": "processing",
            "scaleLevel": "2"
      },
      {
            "aspectId": "webserver",
            "scaleLevel": "3"
      }
    ]
}
```

For information on definition of scale levels, See the VNFD Policies for Scaling.

VNFD Policies for Scaling

There are a number of policies that make up the overall scaling behavior of a VNF. These policies will support the various scaling approaches described above. The first policy defines the aspects that may be scaled (or not):

```
policies:
    - scaling_aspects:
        type: tosca.policies.nfv.ScalingAspects
        properties:
          aspects:
            webserver:
              name: 'webserver'
              description: 'The webserver cluster.'
              max_scale_level: 5
              step_deltas:
                - delta_1
            processing:
              name: 'processing'
              description: 'An example processing function'
              max_scale_level: 3
              step_deltas:
                - delta_1
                - delta_2
                - delta_1
            database:
              name: 'database'
              description: 'A test database'
              max_scale_level: 0
```

You can see in this example that the database aspect has a max_scale_level of 0, which denotes that it cannot be scaled out - this does not mean 0 instances of that aspect - see the algorithm below to see why. The webserver aspect only has a single step_delta, meaning that all scaling steps are uniform whereas the processing aspect has different step_deltas specified for each scaling step. This is called non-uniform scaling. This is only the

declaration of the aspects of this VNF, and this is one of the policies used to perform the validation when a scaling request is received.

Next, they must be applied to VNFCs to control their behavior:

```
- db_initial_delta:
        type: tosca.policies.nfv.VduInitialDelta
        properties:
          initial_delta:
            number_of_instances: 1
        targets: [ vdu1 ]

    - ws_initial_delta:
        type: tosca.policies.nfv.VduInitialDelta
        properties:
          initial_delta:
            number_of_instances: 1
        targets: [ vdu2, vdu4 ]

    - pc_initial_delta:
        type: tosca.policies.nfv.VduInitialDelta
        properties:
          initial_delta:
            number_of_instances: 1
        targets: [ vdu3 ]

    - ws_scaling_aspect_deltas:
        type: tosca.policies.nfv.VduScalingAspectDeltas
        properties:
          aspect: webserver
          deltas:
            delta_1:
              number_of_instances: 1
        targets: [ vdu2, vdu4 ]

    - pc_scaling_aspect_deltas:
        type: tosca.policies.nfv.VduScalingAspectDeltas
        properties:
          aspect: processing
          deltas:
            delta_1:
              number_of_instances: 1
            delta_2:
              number_of_instances: 2
        targets: [ vdu2, vdu4 ]
```

In the examples above, the VNFCs are identified as targets; the aspects could have different behaviours on different VNFCS, but this is not shown here. The definition of the step_deltas are also shown here which are used in the validation and generation of scaling requests (these steps are inferred by the scale level requested). The minimum number of instances of a VNFC is always assumed to be 0 and the maximum number is calculated by the following algorithm:

initial_delta plus the number of instances for each step up to the max_scale_level.

These policies are considered for the scale-level based scaling. There are similar constructs used for instantiation-level based scaling.

```
- instantiation_levels:
        type: tosca.policies.nfv.InstantiationLevels
        properties:
          levels:
            default:
              description: 'Default instantiation level'
              scale_info:
```

```
            database:
              scale_level: 0
            webserver:
              scale_level: 0
            processing:
              scale_level: 0
        premium:
          description: 'Premium instantiation level'
          scale_info:
            database:
              scale_level: 0
            webserver:
              scale_level: 2
            processing:
              scale_level: 3
      default_level: default
```

Similar to the scaling aspects, the first part of the definition of instantiation levels is just their declaration. Here each aspect must already be declared and then each aspect's scale_level is declared for the instantiation level; a default instantiation level is also stipulated in the event that no other is specified. What each scale_level means for each VNFC is further elaborated upon in the VduInstantiationLevels policies, for example:

```
- ws_instantiation_levels:
        type: tosca.policies.nfv.VduInstantiationLevels
        properties:
          levels:
            default:
              number_of_instances: 1
        targets: [ vdu2, vdu4 ]
```

So these policies together state that the default instantiation level is 'default' which will result in the webserver aspect being instantiated at scale_level 0 which is 1 VNFC instance.

Dependencies on Multiple IP Addresses

Static IP Addresses

If the VNFC has connection points configured with a static IP address, the VNFC cannot be scaled as there are no further IP addresses to assign to the connection points on the newly spun up VNFC instances. Instead, a pool of further static IP addresses can be specified. This is an extension to the ETSI specification.

The following example explains how to create a static IP pool using a list of IP addresses, IP ranges or a gateway with netmask (one or a combination of more than one can be specified):

```
vdu2:
  type: cisco.nodes.nfv.Vdu.Compute
  properties:
    name: 'Webserver1'
    description: 'Webserver VNFC'
    vdu_profile:
      min_number_of_instances: 1
      max_number_of_instances: 6
      static_ip_address_pool:
        network: network1
        ip_addresses:
          - ip_address: 192.168.100.0
          - ip_address: 192.168.100.1
          - ip_address: 192.168.100.2
          - ip_address: 192.168.100.3
        ip_address_range:
          - start: 172.16.233.10
```

```
            end: 172.16.233.15
          - start: 172.16.233.20
            end: 172.16.233.25
        gateway: 172.10.11.0
        netmask: 255.255.255.0
```

The scaled out VNFC instance that has connection points with static IP addresses is assigned to a network. This is the key to identify which IP address pool to use when the scaled out instance is deployed. The static IPs are specified at deployment as part of the inputs in the InstantiateVnfRequest. For information on instantiating VNFs, see Instantiating VNFs.

The inputs are provided as part of the additionalParams through the VNFD.

Day Zero Configuration

After deploying the VNFs, day 0 variables are configured in the VNFC instance for the deployment service. In most cases, the values for the day 0 configuration is constant. In other cases, there is a resource pool of values supplied to the day 0 parameter to allow new values to be assigned to the new VNFC instances.

Day 0 configuration within the vendor_section of the VNFD:

```
    vdu3:
      type: cisco.nodes.nfv.Vdu.Compute
      properties:
        name: 'Processing1'
        description: 'Processing VNFC'
        vdu_profile:
          min_number_of_instances: 1
          max_number_of_instances: 5
        vendor_section:
          cisco_esc:
            config_data:
              '/tmp/OSRESTTestETSIDay0_Inline_data.cfg':
                data: |
                  NODE_NAME $NODE_NAME
                  NUM_OF_CPU $NUM_OF_CPU
                  MEM_SIZE $MEM_SIZE
                  PROXY_ADDRS $PROXY_ADDRS
                  SPECIAL_CHARS $SPECIAL_CHARS
                variables:
                  NODE_NAME: vdu_node_1
                  NUM_OF_CPU: 1
                  MEM_SIZE: 1GB
                  PROXY_ADDRS: ["1.1.1.1", "1.1.2.1", "1.1.3.1", "1.1.4.1", "1.1.5.1",
"1.1.6.1", "1.1.7.1"]
                  SPECIAL_CHARS: '`~!@#$%^&*()-_=+[{]}|;:<.>/?'
```

In the above example the day 0 configuration is specified inline, with velocity variables defined in the target configuration. Each of these variables are supported by a variable with one or more values. In order to support multiple values for the $PROXY_ADDRS variable, a list of values are provided. These values are used to populate subsequent uses of the variable on new instances of the VNFC.

For information on day 0 configuration in the deployment data model, see Day Zero Configuration in the *Cisco Elastic Services Controller User Guide*.

Autoscaling of VNFs

KPIs, rules and actions defined in the VNFD determine the conditions under which scaling must be considered. The details are provided in Monitoring Virtual Network Functions. The scaling policies are also defined in

the VNFD using several policy types that control the allowed scaling boundaries. These policy items are described below.

After deployment, ESC configures a monitoring agent (this may be the centralised or distributed instance) with the KPIs to monitor each VNFC. The scaling workflow begins if a KPI reaches its threshold; based on the action defined, ESC performs scale in or scale out and generates appropriate notifications and event logs. This is subject to some built-in functions that can be specified such as `log` or an onboarded script.

ESC sends appropriate notifications to the subscribed consumers. At this time, ESC interrogates the VNF instance resource for the *isAutoscaleEnabled* flag (this is set initially by the value in the VNFD but can be modified after creation). If this flag is set to true, ESC invokes the scaling workflow (instigated using a *ScaleVnfToLevelRequest* to request the scaling of multiple aspects in a single request). If the *isAutoscaleEnabled* is set to false, then the control is with an external system such as an NFVO or EM to trigger the desired action using the requests described above.

Note While creating an auto scaling or auto healing request, any new external requests are blocked. The user is notified of the corresponding response and problem details of the blocked request.